Polymorphism, Subtyping, and Type Inference in MLsub

Stephen Dolan and Alan Mycroft

January 18, 2017

Computer Laboratory
University of Cambridge
The select function

\[
\text{select } p \lor v \downarrow d = \text{if } (p \lor v) \text{ then } v \text{ else } d
\]

In ML, select has type scheme

\[
\forall \alpha. (\alpha \to \text{bool}) \to \alpha \to \alpha \to \alpha
\]
Data flow in select

\[ \text{select } p \lor d = \text{if } (p \lor) \text{ then } v \text{ else } d \]

- \( v \) argument to \( p \)
- \( d \) result

In MLsub, select has this type scheme:

\[ \forall \alpha, \beta. (\alpha \rightarrow \text{bool}) \rightarrow \alpha \rightarrow \beta \rightarrow (\alpha \sqcup \beta) \]
Data flow in select

\[ \text{select } p \lor d = \text{if } (p \lor) \text{ then } \lor \text{ else } d \]

In MLsub, select has this type scheme:

\[ \forall \alpha, \beta. (\alpha \rightarrow \text{bool}) \rightarrow \alpha \rightarrow \beta \rightarrow (\alpha \sqcup \beta) \]
The MLsub Type System
\Gamma \vdash e : \tau
Γ ⊢ e : τ
Expressions of MLsub

We have functions

\[ x \quad \lambda x.e \quad e_1 \quad e_2 \]

... and records

\[ \{ \ell_1 = e_1, \ldots, \ell_n = e_n \} \quad e.\ell \]

... and booleans

\[ \text{true} \quad \text{false} \quad \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \]

... and let

\[ \hat{x} \quad \text{let } \hat{x} = e_1 \text{ in } e_2 \]
\[ \Gamma \vdash e : \tau \]
Typing rules of MLsub

MLsub is

\[
\begin{align*}
\text{(SUB)} & \quad \frac{\Gamma \vdash e : \tau_1 \quad \tau_1 \leq \tau_2}{\Gamma \vdash e : \tau_2}
\end{align*}
\]
Γ ⊢ e : τ
Constructing Types

The standard definition of types looks like:

$$\tau ::= \bot \mid \tau \rightarrow \tau \mid \top$$

(ignoring records and booleans for now)
Constructing Types

The standard definition of types looks like:

\[ \tau ::= \bot \mid \tau \to \tau \mid \top \]

(ignoring records and booleans for now)

with a subtyping relation like:

\[ \bot \leq \tau \quad \tau \leq \top \quad \frac{\tau_1' \leq \tau_1 \quad \tau_2 \leq \tau_2'}{\tau_1 \to \tau_2 \leq \tau_1' \to \tau_2'} \]
Lattices

These types form a lattice:

- least upper bounds $\tau_1 \sqcup \tau_2$
- greatest lower bounds $\tau_1 \sqcap \tau_2$
These types form a lattice:

- least upper bounds $\tau_1 \sqcup \tau_2$
- greatest lower bounds $\tau_1 \sqcap \tau_2$

\[
\begin{array}{c}
e_1 : \tau_1 \\
e_2 : \tau_2 \\
\hline
\text{if rand () then } e_1 \text{ else } e_2 : \tau_1 \sqcup \tau_2
\end{array}
\]
Bizzarely difficult questions

Is this true, for all $\alpha$?

$$\alpha \rightarrow \alpha \leq \bot \rightarrow T$$
Bizzarely difficult questions

Is this true, for all $\alpha$?

$$\alpha \to \alpha \leq \bot \to \top$$

How about this?

$$(\bot \to \top) \to \bot \leq (\alpha \to \bot) \sqcup \alpha$$

Yes, it turns out, by case analysis on $\alpha$. And only by case analysis.
Bizzarely difficult questions

Is this true, for all $\alpha$?

$$\alpha \rightarrow \alpha \leq \bot \rightarrow T$$

How about this?

$$(\bot \rightarrow T) \rightarrow \bot \leq (\alpha \rightarrow \bot) \sqcup \alpha$$

Yes, it turns out, by case analysis on $\alpha$. 
Bizzarely difficult questions

Is this true, for all $\alpha$?

$$\alpha \rightarrow \alpha \leq \bot \rightarrow \top$$

How about this?

$$\left(\bot \rightarrow \top\right) \rightarrow \bot \leq (\alpha \rightarrow \bot) \sqcup \alpha$$

Yes, it turns out, by case analysis on $\alpha$.

And only by case analysis.
Extensibility

Let’s add a new type of function $\tau_1 \circ \rightarrow \tau_2$. Now we have a counterexample: $\alpha = (\top \circ \rightarrow \bot) \circ \rightarrow \bot$. 
Extensibility

Let’s add a new type of function $\tau_1 \circ \tau_2$. It’s a supertype of $\tau_1 \to \tau_2$

“function that may have side effects”
Let’s add a new type of function $\tau_1 \xrightarrow{\circ} \tau_2$. It’s a supertype of $\tau_1 \rightarrow \tau_2$

“function that may have side effects”

Now we have a counterexample:

$$\alpha = (\top \xrightarrow{\circ} \bot) \xrightarrow{\circ} \bot$$
Extensibility and vacuous reasoning

\[ \tau_r = \text{some record type} \]
\[ \tau_f = \text{some function type} \]

Is \( \tau_r \sqcap \tau_f \leq \text{bool?} \)
Extensibility and vacuous reasoning

\[ \tau_r = \text{some record type} \]
\[ \tau_f = \text{some function type} \]

Is \( \tau_r \cap \tau_f \leq \text{bool} \)?

Yes, vacuously.
Two techniques give us an extensible system:

- Add explicit type variables as indeterminates

gets rid of case analysis
Two techniques give us an extensible system:

- Add explicit type variables as indeterminates
  
  *gets rid of case analysis*

- Require a distributive lattice
  
  *gets rid of vacuous reasoning*
We end up with all the standard types
Resulting types

We end up with all the standard types

... with the same subtyping order
Resulting types

We end up with all the standard types
... with the same subtyping order
... but we identify fewer of the weird types

\{\text{foo : bool}\} \sqcap (\top \rightarrow \top) \not\leq \text{bool}
Type Inference with Polar Types
Input and output types

$\tau \sqcup \tau'$: produces a value which is a $\tau$ or a $\tau'$

$\tau \sqcap \tau'$: requires a value which is a $\tau$ and a $\tau'$

$\sqcup$ is for outputs, and $\sqcap$ is for inputs.
Input and output types

\( \tau \uplus \tau' \): produces a value which is a \( \tau \) or a \( \tau' \)

\( \tau \cap \tau' \): requires a value which is a \( \tau \) and a \( \tau' \)

\( \uplus \) is for outputs, and \( \cap \) is for inputs.

Divide types into

- output types \( \tau^+ \)
- input types \( \tau^- \)
Polar types

\[ \tau^+ ::= \text{bool} \mid \tau_1^- \to \tau_2^+ \mid \{ \ell_1 : \tau_1^+, \ldots, \ell_n : \tau_n^+ \} \mid \alpha \mid \tau_1^+ \sqcup \tau_2^+ \mid \bot \mid \mu \alpha. \tau^+ \]

\[ \tau^- ::= \text{bool} \mid \tau_1^+ \to \tau_2^- \mid \{ \ell_1 : \tau_1^-, \ldots, \ell_n : \tau_n^- \} \mid \alpha \mid \tau_1^- \sqcap \tau_2^- \mid \top \mid \mu \alpha. \tau^- \]
Cases of unification

In HM inference, unification happens in three situations:

- Unifying two input types
- Unifying two output types
- Using the output of one expression as input to another
Cases of unification

In HM inference, unification happens in three situations:

- Unifying two input types
  \( \text{Introduce} \ □ \)

- Unifying two output types
  \( \text{Introduce} \ □ \)

- Using the output of one expression as input to another
  \( \tau^+ \leq \tau^- \) constraint
Suppose we have an identity function

\[ \alpha \rightarrow \alpha \]
Suppose we have an identity function, which uses its argument as a $\tau$

$$\alpha \rightarrow \alpha \mid \alpha = \tau$$
Suppose we have an identity function, which uses its argument as a $\tau$

\[
\alpha \rightarrow \alpha \mid \alpha = \tau
\]

\[
\equiv \forall \tau \rightarrow \tau
\]
Suppose we have an identity function, which uses its argument as a $\tau$

$$\alpha \rightarrow \alpha \mid \alpha = \tau$$

$$\equiv^{\forall} \tau \rightarrow \tau$$

The substitution $[\tau/\alpha]$ solves the constraint $\alpha = \tau$
Suppose we have an identity function, which uses its argument as a $\tau^-$. 

$$\alpha \rightarrow \alpha \mid \alpha \leq \tau^-$$
Suppose we have an identity function, which uses its argument as a $\tau^-$. 

$$\alpha \rightarrow \alpha \mid \alpha \leq \tau^-$$

$$\equiv \forall (\alpha \cap \tau^-) \rightarrow \alpha$$
Suppose we have an identity function, which uses its argument as a $\tau^-$. 

$$\alpha \rightarrow \alpha \mid \alpha \leq \tau^-$$

$$\equiv \forall (\alpha \sqcap \tau^-) \rightarrow \alpha$$

The *bisubstitution* $[\alpha \sqcap \tau^-/\alpha^-]$ solves $\alpha \leq \tau^-$
Decomposing constraints

We only need to decompose constraints of the form \( \tau^+ \leq \tau^- \).

\[
\tau_1 \sqcup \tau_2 \leq \tau_3 \quad \equiv \quad \tau_1 \leq \tau_3, \quad \tau_2 \leq \tau_3 \\
\tau_1 \leq \tau_2 \sqcap \tau_3 \quad \equiv \quad \tau_1 \leq \tau_2, \quad \tau_1 \leq \tau_3
\]

Thanks to the input/output type distinction, the hard cases of \( \tau_1 \sqcap \tau_2 \leq \tau_3 \) and \( \tau_1 \leq \tau_2 \sqcup \tau_3 \) can never come up.
Combining solutions

We solve a system of multiple constraints $C_1, C_2$ by:

- Solving $C_1$, giving a bisubstitution $\xi$
- Applying that to $C_2$
- Solving $\xi C_2$, giving a bisubstitution $\zeta$

Then $\xi \circ \zeta$ solves the system $C_1, C_2$. 
MLsub infers types by walking the syntax of the program, but must deal with subtyping constraints rather than just equalities. Thanks to:

- algebraically well-behaved types
- polar types, restricting occurrences of $\sqcup$ and $\sqcap$
- the biunification algorithm

we can always handle these constraints, producing a principal type.
Future work

{-# LANGUAGE ?? #-}

- RankNTypes?
- GADTs?
- TypeFamilies?
- ...

27
Questions?

http://www.cl.cam.ac.uk/~sd601/mlsub
stephen.dolan@cl.cam.ac.uk
Mutable references

References are generally considered “invariant”.

Instead, consider \( \text{ref} \) a two-argument constructor

\[
(\alpha, \beta) \text{ ref}
\]

with operations:

\[
\begin{align*}
\text{make} &: (\alpha, \alpha) \text{ ref} \\
\text{get} &: (\bot, \beta) \text{ ref} \rightarrow \beta \\
\text{set} &: (\alpha, \top) \text{ ref} \rightarrow \alpha \rightarrow \text{unit}
\end{align*}
\]